Towards Exact State Complexity Bounds for Input-Driven Pushdown Automata

Galina Jirásková[1(✉)] and Alexander Okhotin[2]

[1] Mathematical Institute, Slovak Academy of Sciences,
Grešákova 6, 040 01 Košice, Slovakia
jiraskov@saske.sk
[2] St. Petersburg State University,
7/9 Universitetskaya nab., Saint Petersburg 199034, Russia
alexander.okhotin@spbu.ru

Abstract. The paper improves several state complexity bounds for input-driven pushdown automata (IDPDA), also known as visibly pushdown automata. For deterministic IDPDA it is proved that the number of states sufficient and in the worst case necessary to represent the reversal of an n-state automaton is exactly n^n if all inputs are assumed to be well-nested, and between n^n and $n(n^n - (n-1)^n) + 1$ without this restriction, cf. $2^{\Theta(n \log n)}$ in the literature. For the concatenation of an m-state and an n-state IDPDA, the new lower bound is mn^n, which is asymptotically tight for well-nested inputs. Without this restriction, the state complexity is between mn^n and $m(n+1)n^n 2^n$. Finally, it is proved that transforming an n-state nondeterministic IDPDA to a deterministic one requires exactly 2^{n^2} states, cf. $2^{\Theta(n^2)}$ in the literature; the known lower bounds on complementing nondeterministic IDPDA and on transforming them to unambiguous are also improved.

1 Introduction

Input-driven pushdown automata (IDPDA), first studied by Mehlhorn [5] and by von Braunmühl and Verbeek [2], are a special case of deterministic pushdown automata, in which the input alphabet is split into three disjoint classes, so that the type of a symbol determines the action on the stack that the automaton must perform: on *left brackets* it may only push one symbol onto the stack; on *right brackets* it may only pop one stack symbol; on *neutral symbols* the stack may not be accessed. The model was later reintroduced by Alur and Madhusudan [1] under the name of "visibly pushdown automata", which caused a revival of research on this model.

Input-driven automata can be defined in both deterministic (DIDPDA) and nondeterministic (NIDPDA) variants, which were proved to be equivalent by von Braunmühl and Verbeek [2]: an n-state NIDPDA can be transformed to a

G. Jirásková—Research supported by VEGA grant 2/0084/15 and grant APVV-15-0091.

M. Hoshi and S. Seki (Eds.): DLT 2018, LNCS 11088, pp. 441–452, 2018.
https://doi.org/10.1007/978-3-319-98654-8_36

Table 1. Known lower and upper bounds on the state complexity of basic operations for deterministic input-driven automata.

	Well-nested inputs		Arbitrary inputs	
	Lower bound	Upper bound	Lower bound	Upper bound
$\cup$	$\Theta(mn)$ [12]	mn [1]	$\Theta(mn)$ [12]	mn [1]
$\cap$	$\Theta(mn)$ [12]	mn [1]	$\Theta(mn)$ [12]	mn [1]
$\sim$	n	n	n	n
$\cdot$	$m2^{\Theta(n \log n)}$ [8]	$m(n^n + 2^n)$ [8]	$m2^{\Theta(n \log n)}$ [8]	$m \cdot n^n \cdot 4^n$ [8]
^{2}	$2^{\Theta(n \log n)}$ [8]	$n(n^n + 2^n)$ [8]	$2^{\Theta(n \log n)}$ [8]	$n \cdot n^n \cdot 4^n$ [8]
*	$2^{\Theta(n \log n)}$ [13]	$n^n + 2^n + 1$ [8]	$2^{\Theta(n \log n)}$ [13]	$n^n \cdot 4^n$ [8]
^{R}	$2^{\Omega(n \log n)}$ [12]	n^n [8]	$2^{\Omega(n \log n)}$ [12]	$n^n \cdot 2^n$ [8]

2^{n^2}-state DIDPDA by an elaborated form of the subset construction; later, Alur and Madhusudan [1] established a lower bound of $2^{\Omega(n^2)}$ states. An intermediate model with unambiguous nondeterminism (UIDPDA) was introduced by Okhotin and Salomaa [7], who showed that transforming UIDPDA to DIDPDA, as well as NIDPDA to UIDPDA, requires $2^{\Omega(n^2)}$ states. A few generalizations of input-driven automata with strictly greater expressive power have been studied, such as the transducer-driven automata [3].

The family of languages recognized by input-driven automata is notable for being closed under most of the standard operations on formal languages. The closure results were first systematically studied by Alur and Madhusudan [1], whose work inspired the state complexity studies for this model. Piao and Salomaa [12] and Salomaa [13] proved the first lower bounds on the state complexity of basic operations on DIDPDA. Later Okhotin and Salomaa [8] presented the first efficient constructions for concatenation, Kleene star and reversal, with the number of states close to the known lower bounds. Notably, the constructions produce fewer states if the automata are assumed to operate only on well-nested strings. The currently known bounds are optimal up to a constant factor in the exponent, and are presented in Table 1, where the cases of automata operating only on well-nested inputs, and of unrestricted automata, are presented separately.

Besides these basic operations on languages, the state complexity of some further operations on DIDPDA has been studied. There are recent results on the number of states in DIDPDA for the quotient [10], the edit distance neighborhood [9], insertion, deletion, square root and proportional removals [11].

The purpose of this paper is to improve both lower and upper bounds on the state complexity of basic operations on DIDPDA, as well as on the transformations between input-driven models. The paper begins with the results on the reversal operation, presented in Sect. 3. First, an upper bound of $n(n^n-(n-1)^n)+1$ states is established by a new construction of an input-driven automaton recognizing the reversal. At the same time, an asymptotically tight

lower bound of n^n states is presented, which uses only well-nested strings. This matches the upper bound of n^n states in the well-nested case by Okhotin and Salomaa [8].

In Sect. 4, new bounds are established for concatenation. First, there is a new lower bound of mn^n states in the well-nested case, which is asymptotically tight to the known upper bound of $m(n^n + 2^n)$ states [8]. Second, an upper bound of $m(n+1)n^n 2^n$ states is sketched for the general case, improving the construction by Okhotin and Salomaa [8]. For the Kleene star and for the square, there is a new lower bounds of n^n states in each case, which is asymptotically tight.

The last contribution, presented in Sect. 5, is several improved lower bounds on nondeterministic input-driven automata (NIDPDA). For the transformation of an n-state nondeterministic IDPDA to a deterministic one, it is shown that 2^{n^2} states are needed in the worst case, matching the upper bound [2] and improving over a known lower bound of $2^{\Theta(n^2)}$ states [1]. The same construction is used to improve the known bounds on transforming NIDPDA to UIDPDA and on complementing NIDPDA [7].

2 Deterministic Input-Driven Automata

In an input-driven pushdown automaton (IDPDA), the input alphabet is split into three disjoint classes, Σ_{+1}, Σ_{-1} and Σ_0, and the type of a symbol determines the action on the stack that the automaton must perform upon reading that symbol. The symbols in Σ_{+1} are known as the *left brackets* (notation: $<$, $\ll$): on these symbols, the automaton must push one symbol onto the stack and cannot examine the stack. The symbols in Σ_{-1} are *right brackets* (notation: $>$, $\gg$), on which the automaton pops one symbol from the stack. Finally, on any *neutral symbols* in Σ_0, the automaton cannot access the stack at all, and behaves like a finite automaton.

By the original definition of input-driven automata used by Mehlhorn [5] and by von Braunmühl and Verbeek [2], IDPDA are assumed to operate on well-nested strings. The definition by Alur and Madhusudan [1] further allows ill-nested strings: on unmatched left brackets, an automaton pushes symbols that shall never be popped; on unmatched right brackets, it uses special transitions *by an empty stack*, in which the automaton detects the stack emptiness and leaves the stack empty.

Definition 1 (Mehlhorn [5], Alur and Madhusudan [1]). *A (deterministic) input-driven pushdown automaton (IDPDA) consists of the following components.*

- *The input alphabet Σ is a finite set split into three disjoint classes: $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$.*
- *The set of (internal) states Q is a finite set, with an initial state $q_0 \in Q$ and with a subset of accepting states $F \subseteq Q$.*
- *The stack alphabet Γ is a finite set, and a special symbol $\bot \notin \Gamma$ is used to denote an empty stack.*

- *For each neutral symbol $c \in \Sigma_0$, the transitions by this symbol are described by a function $\delta_c \colon Q \to Q$.*
- *For each left bracket symbol $< \in \Sigma_{+1}$, the behaviour of the automaton is described by functions $\delta_< \colon Q \to Q$ and $\gamma_< \colon Q \to \Gamma$, which, for a given current state, provide the next state and the symbol to be pushed onto the stack, respectively.*
- *For every right bracket symbol $> \in \Sigma_{-1}$, there is a function $\delta_> \colon Q \times (\Gamma \cup \{\bot\}) \to Q$ specifying the next state, assuming that the given stack symbol is popped from the stack (or that the stack is empty).*

A triple (q, w, x), in which $q \in Q$ is the current state, $w \in \Sigma^*$ is the remaining input and $x \in \Gamma^*$ is the stack contents, is called a configuration. *For each configuration, the next configuration is defined as follows.*

$$\begin{aligned}
(q, cw, x) &\vdash (\delta_c(q), w, x) && (c \in \Sigma_0,\ q \in Q)\\
(q, <w, x) &\vdash (\delta_<(q), w, \gamma_<(q)x), && (< \in \Sigma_{+1},\ q \in Q)\\
(q, >w, sx) &\vdash (\delta_>(q, s), w, x) && (> \in \Sigma_{-1},\ q \in Q,\ s \in \Gamma)\\
(q, >w, \varepsilon) &\vdash (\delta_>(q, \bot), w, \varepsilon) && (> \in \Sigma_{-1})
\end{aligned}$$

In all cases, $w \in \Sigma^*$ is the remaining input and $x \in \Gamma^*$ is a string of symbols in the stack.

The language recognized by A is the set of all strings $w \in \Sigma^*$, on which the automaton, having begun its computation in the configuration (q_0, w, ε), eventually reaches a configuration of the form (q, ε, x), with $q \in F$ and with any stack contents $x \in \Gamma^*$.

When an input-driven automaton reads a left bracket $< \in \Sigma_{+1}$, it pushes a symbol onto the stack. This symbol is popped at the exact moment when the automaton encounters the matching right bracket $> \in \Sigma_{-1}$. Thus, a computation of an input-driven automaton on any well-nested substring leaves the original stack contents untouched and unexamined. Accordingly, the behaviour of an IDPDA with a set of states Q on a well-nested substring w is characterized by a function $f \colon Q \to Q$, where $f(q)$ is the state in the end of the computation on w that began in the state q. One of the basic constructions of IDPDA, which is crucial for recognizing concatenation, Kleene star and reversal, is an IDPDA that *computes the behaviour function* of a given IDPDA. Denote $Q^Q = \{f \mid f \colon Q \to Q\}$.

Lemma A (Okhotin and Salomaa [8]). *For every IDPDA A with the set of states Q and the stack alphabet Γ, there exists an IDPDA C with the set of states Q^Q and the stack alphabet $Q^Q \times \Sigma_{+1}$, that calculates the behaviour of A on the longest well-nested suffix of the read portion of the input.*

3 Reversal

The *reversal* of a string $w = a_1 \dots a_n$, with $a_i \in \Sigma$, is the same string written backwards: $w^R = a_n \dots a_1$; the reversal of a language is defined element-wise:

$L^R = \{w^R \mid w \in L\}$. In the case of input-driven automata, the reversal of a language also implies that left and right brackets exchange their roles, so that the reversal of a string over an alphabet $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ becomes a string over the alphabet $(\Sigma_{-1}, \Sigma_{+1}, \Sigma_0)$. The reversal of a well-nested string is well-nested.

Under this assumption, the family of input-driven languages is closed under the reversal [1], and the reversal of an n-state IDPDA is representable by an IDPDA with $n^n \cdot 2^n$ states [8]. This result is achieved by the following construction.

Lemma B (Okhotin and Salomaa [8], Lemma 11). *For every IDPDA A over an alphabet $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ with a set of states Q and a stack alphabet Γ, there exists an IDPDA C over the inverted alphabet $(\Sigma_{-1}, \Sigma_{+1}, \Sigma_0)$ with the set of states $Q^Q \times 2^Q$ and the stack alphabet $Q^Q \times 2^Q \times \Sigma_{-1}$ that recognizes the language $L(A)^R$.*

Let uv be a string, where u is its longest well-nested prefix. Then, the state of C entered upon reading $(uv)^R$ is a pair (f, S), where $f\colon Q \to Q$ is the behaviour of A on u, and $S \subseteq Q$ is the set of all states of A, from which it would accept uv beginning with the empty stack.

Proof (sketch of the construction). The initial state is (id, F), where F is the set of accepting states of A. Upon reading a neutral symbol $c \in \Sigma_0$, the automaton C composes the behaviour function with the behaviour on c and replaces S with its pre-images under transitions on c.

$$\delta'_c\big((f, S)\big) = \big(f \circ \delta_c,\ \delta_c^{-1}(S)\big).$$

Upon reading a right bracket ($> \in \Sigma_{-1}$), which it treats as a left bracket, C pushes the bracket and the current values of f and S onto the stack, and begins computing a new behaviour function inside the brackets.

$$\delta'_>\big((f, S)\big) = (id,\ \{q \mid \delta_>(q, \bot) \in S\})$$
$$\gamma'_>\big((f, S)\big) = (f, S, >)$$

Upon reading a left bracket ($< \in \Sigma_{-1}$), which is a right bracket for C, it reconstructs the behaviour on the well-nested substring ending at this bracket, composes it with the behaviour function popped from the stack and continues the simulation on the current level of brackets.

Since the lower bound in Theorem 4 applies, the state complexity of reversal for IDPDA on arbitrary inputs is at least n^n and at most $n^n \cdot 2^n$, and the question is, what is its exact state complexity? Can the construction in Lemma B be improved to produce fewer states?

The first, easy observation on this construction is that not all states of the form (f, S) are reachable. Indeed, if $f(i) = f(j)$ for any distinct states i and j, this means that the computations beginning in i and in j with the empty stack coincide, and therefore either both are accepting or both are rejecting, that is, S contains both or neither. If this condition does not hold, then the pair (f, S) can be excluded from the set of states.

A more substantial improvement to this construction is based on the following two observations. **First**, as the simulation progresses on the same level of brackets, for the consecutive pairs (f_0, S_0), (f_1, S_1), (f_2, S_2), etc. encountered on this level, the sets $f_i(S_i)$ form a chain of containment, and the sets $f_i(\overline{S_i})$ form another chain of containment.

$$f_0(S_0) \supseteq f_1(S_1) \supseteq f_2(S_2) \supseteq \dots$$
$$f_0(\overline{S_0}) \supseteq f_1(\overline{S_1}) \supseteq f_2(\overline{S_2}) \supseteq \dots$$

Suppose that the first pair (f_0, S_0) on some level of brackets satisfies the condition that *all states in* $f_0(S_0)$ *are less than all states in* $f_0(\overline{S_0})$, under some fixed linear order on Q. Then, this property becomes an *invariant*, satisfied by all pairs on this level of brackets. The construction in Lemma B, of course, does not guarantee that this condition would ever hold, and the question is, can the construction be modified to make it hold?

The **second** observation is that, while the behaviour function on some level of brackets is being constructed, its value is not yet used: the automaton only composes it with the behaviour functions on the neutral symbols and well-nested substrings it encounters. For that reason, it is not imperative that the correct value of the behaviour function is remembered in the course of this computation. At the moment when the automaton enters a new level of brackets, it need not necessarily enter the state (id, S), but may actually enter a state of the form (π, S), where π is any permutation, as long as the same permutation will be available to the automaton when it eventually exits this level of brackets. Then, if the actual behaviour function on the string u, as in Lemma B, is φ, and the set is S, the automaton shall actually remember a pair $(\pi \circ \varphi, S)$. When it reads the matching bracket, it composes the inverse π^{-1} of this permutation with the computed function $\pi \circ \varphi$, and obtains the behaviour function π.

The purpose of doing this is in choosing the function π in order to satisfy the condition that *all states in* $\pi(S)$ *are less than all states in* $\pi(\overline{S})$. Then the set of states can be limited to the states satisfying this condition.

Lemma 2. *For every IDPDA A over an alphabet $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ with a set of states Q and a stack alphabet Γ, there exists an IDPDA C over the inverted alphabet $(\Sigma_{-1}, \Sigma_{+1}, \Sigma_0)$ with the following set of states Q' and stack alphabet Γ', that recognizes the language $L(A)^R$.*

$$Q' = \{(f, S) \mid f\colon Q \to Q,\ S \subseteq Q,\ S \neq \varnothing,\ \max f(S) < \min f(\overline{S})\} \cup \{q_{dead}\}$$
$$\Gamma' = \{(f, S, >) \mid f\colon Q \to Q,\ S \subseteq Q,\ S \neq \varnothing,\ \max f(S) < \min f(\overline{S}),\ > \in \Sigma_{-1}\}$$

Let uv be a string, where u is its longest well-nested prefix, and let π be a permutation that functionally depends on the set of states, from which A would accept v beginning with the empty stack. Then, the state of C entered upon reading $(uv)^R$ is a pair $(\pi \circ \varphi, S)$, where $\varphi\colon Q \to Q$ is the behaviour of A on u, and $S \subseteq Q$ is the set of all states of A, from which it would accept uv beginning with the empty stack.

Proof (sketch of the construction). Let $Q = \{0, \ldots, n-1\}$. The initial state is $(\pi_F,\ F)$, where π_F is chosen to have $\pi_F(F) = \{0, \ldots, |F| - 1\}$.

The transitions by neutral symbols are the same as in Lemma B,

When C reads a right bracket ($> \in \Sigma_{-1}$) in a state (f, S), it acts as follows. Let $\widehat{S}$ be the set of all states q with $\delta_>(q, \bot) \in S$, and let π be a permutation with $\pi(\widehat{S}) = \{0, \ldots, |\widehat{S}| - 1\}$. Then the automaton pushes the context of the current simulation onto the stack as in Lemma B, and uses π inside the brackets instead of the identity behaviour function.

$$\delta'_>\big((f, S)\big) = (\pi,\ \widehat{S}) \qquad\qquad \gamma'_>\big((f, S)\big) = (f, S, >)$$

Upon reading a left bracket ($< \in \Sigma_{-1}$), the automaton reconstructs π from f, S and the bracket ($>$), and uses it to reconstruct the behaviour on the well-nested substring. If S ever becomes the empty set, the automaton enters the state q_{dead} and stay therein until the end of the computation. □

It remains to estimate the number of states in the automaton constructed in Lemma 2, which is done in the following theorem.

Theorem 3. *The reversal of an n-state IDPDA over any stack alphabet can be recognized by an IDPDA with $n(n^n - (n-1)^n) + 1$ states, using $n(n^n - (n-1)^n) \cdot |\Sigma_{-1}|$ stack symbols.*

Proof. In the automaton constructed in Lemma 2, states are pairs (f, S), with $f\colon Q \to Q$, $S \subseteq Q$ and $S \neq \varnothing$ satisfying $\max f(S) < \min f(\overline{S})$ Every such pair (f, S) characterized by a pair (f, i), where $i = \max(f(S))$. Indeed, (f, i) encodes the pair $\big(f, f^{-1}(\{0, \ldots, i\})\big)$.

How many pairs (f, i) are there? There are n different choices of i, and for each i, out of all n^n functions $f\colon Q \to Q$, there are $(n-1)^n$ functions with $i \notin Im(f)$, which cannot form a pair with i. Altogether, there are exactly $n(n^n - (n-1)^n)$ admissible pairs. □

This upper bound on the state complexity of the reversal is accompanied by the following asymptotically tight lower bound, improving the previously known lower bound $2^{\Omega(n \log n)}$ [12, Theorem 5.1].

Theorem 4. *For every $n \geqslant 3$, there exists an n-state IDPDA A over an alphabet with one left bracket, one right bracket and three neutral symbols, which accepts only well-nested strings, such that every IDPDA recognizing $L(A)^R$ requires at least n^n states.*

Proof. The IDPDA is defined over the alphabet $\Sigma_{+1} = \{<\}$, $\Sigma_{-1} = \{>\}$, $\Sigma_0 = \{a, b, c\}$, with the set of states $Q = \{0, \ldots, n-1\}$, where 0 is the initial and the only accepting state. The same stack alphabet is $\Gamma = Q$.

Transitions by a increment the state by one ($\delta_a(i) = (i+1) \bmod n$), transitions by b exchange states 0 and 1 ($\delta_b(0) = 1$, $\delta_b(1) = 0$, and $\delta_b(i) = i$ for $i \geqslant 2$), transitions by c map 0 and 1 to 0 ($\delta_c(0) = \delta_c(1) = 0$ and $\delta_c(i) = i$ for $i \geqslant 2$). On a left bracket, the automaton pushes the current state and enters the state zero:

$\delta_<(i) = 0$, $\gamma_<(i) = i$. On a right bracket, the automaton verifies whether the state and the stack symbol are the same: $\delta_>(i,i) = 0$ for all i, and $\delta_>(i,j) = 1$ for $i \neq j$. The symbols a, b, c perform three transformations on Q that generate the semigroup of all transformations on Q. For each transformation $f\colon Q \to Q$, let the string $w_f \in \{a,b,c\}^*$ encode its representation as a composition of these three generators.

Let B be any IDPDA recognizing the language $L(A)^R$, the goal is to prove that it must have at least n^n states. For each function $f\colon Q \to Q$, let q_f be the state reached by B upon reading the string $(w_f>)^R$; the stack contents at this point are always the same. It is claimed that these states are pairwise distinct. For every two distinct functions f and g, let $i \in Q$ be an argument, on which they differ: $f(i) \neq g(i)$. Let h be the transposition of 0 and i, and let $v = a^{f(i)}<w_h$. Then A accepts the string $vw_f>$ and rejects the string $vw_g>$, as follows.

$$(0,\bot) \xrightarrow{a^{f(i)}} (f(i),\bot) \xrightarrow{<} (0,f(i)) \xrightarrow{w_h} (i,f(i)) \xrightarrow{w_f} (f(i),f(i)) \xrightarrow{>} (0,\bot),$$

$$(0,\bot) \xrightarrow{a^{f(i)}} (f(i),\bot). \xrightarrow{<} (0,f(i)) \xrightarrow{w_h} (i,f(i)) \xrightarrow{w_g} (g(i),f(i)) \xrightarrow{>} (1,\bot).$$

Therefore, B must accept the string $(vw_f>)^R$ and rejects the string $(vw_g>)^R$. For this to happen, its configurations after reading $(w_f>)^R$ and $(w_g>)^R$ must be distinct. This is only possible if $q_f \neq q_g$. □

For any n-state IDPDA operating only on well-nested inputs, there is a known construction of an IDPDA for its reversal that has only n^n states [8, Lemma 10]. Therefore, Theorem 4 gives a tight bound in this special case.

4 Concatenation and Star

For the concatenation of an m-state IDPDA with an n-state IDPDA, there is a construction of an IDPDA with $m(n^n + 2^n)$ states in the well-nested case, and with $m \cdot n^n \cdot 4^n$ in the general case [8]. The lower bound is $m2^{O(n \log n)}$ [8]. The following improved lower bound asymptotically matches the upper bound in the well-nested case.

Theorem 5 (Concatenation: lower bound). *For every $m, n \geqslant 3$, there exist an m-state IDPDA A and an n-state IDPDA B, both defined over an alphabet with one left bracket, two right brackets and four neutral symbols, and both accepting only well-nested strings, such that every IDPDA for $L(A) \cdot L(B)$ requires at least mn^n states.*

Proof. Let A be an m-state DFA that counts the number of d's modulo m, and B be an n-state IDPDA with the set of states $\{0, 1, \ldots, n-1\}$, with the same stack alphabet, the initial state 0, the unique final state $n-1$, and transitions

$\delta_a(j) = (j+1) \bmod n$;
$\delta_b(0) = 1$, $\delta_b(1) = 0$, and $\delta_b(j) = j$ if $j \geqslant 2$;
$\delta_c(0) = \delta_c(1) = 0$ and $\delta_c(j) = j$ if $j \geqslant 2$;

$\delta_d(j) = j$;
$\delta_<(j) = j$ and $\gamma_<(j) = j$;
$\delta_>(j, \beta) = \beta$ if $j = 0$ and $\delta_>(j, \beta) = 0$ otherwise;
$\delta_\gg(j, \beta) = 1$ if $i = \beta = 0$ and $\delta_\gg(j, \beta) = n - 1$ otherwise.
It can be shown that every IDPDA recognizing $L(A)L(B)$ requires at least mn^n states. □

Turning to the general case, without the well-nestedness requirement, the construction by Okhotin and Salomaa [8] produces a set of states $P \times 2^Q \times 2^Q \times Q^Q$, where P and Q are sets of states of the automata being concatenated. In other words, the constructed IDPDA remembers a state of the first IDPDA, and two subsets of states and a behaviour function of the second IDPDA. The question is, can the idea of Lemma 2 be applied here to reduce the number of states in a similar way?

It turns out that one of the two sets of states remembered in this construction can indeed be "optimized" in the same way as in the case of reversal. Again, a permutation is applied to the behaviour function at each level of brackets, so that this set can be inferred from it. Unfortunately, the other set apparently cannot be handled similarly, because, the way it is updated in the computation, it may occasionally change independently of the behaviour function. Overall, the construction uses states of the following form.

$$Q' = \{(p, f, S, S') \mid p \in P,\ f\colon Q \to Q,\ S, S' \subseteq Q,\ \exists i : S' = \{f(0), \ldots, f(i-1)\}\}$$

Similarly to Lemma 2, instead of the set S', it is then sufficient to store the number i. An upper bound on the number of such states is stated in the following theorem, which somewhat improves on the construction by Okhotin and Salomaa [8].

Theorem 6. *Let A be an m-state and B an n-state IDPDA over the same alphabet. Then, there exists an IDPDA with $m \cdot (n+1) \cdot n^n \cdot 2^n$ states that recognizes the concatenation $L(A) \cdot L(B)$.*

Theorem 7 (Square: lower bound). *For every $n \geqslant 3$, there is an n-state IDPDA A defined over an alphabet with one left bracket, two right brackets and four neutral symbols, and accepting only well-nested strings, such that every IDPDA for $L(A)^2$ requires at least n^n states.*

The proof of Theorem 7 works for the Kleene star as well, and the following lower bound is established.

Theorem 8 (Star: lower bound). *For every $n \geqslant 3$, there is an n-state IDPDA A defined over an alphabet with one left bracket, two right brackets and four neutral symbols, and accepting only well-nested strings, such that every IDPDA for $L(A)^*$ requires at least n^n states.*

5 Improved Bounds for Nondeterministic Automata

A *nondeterministic input-driven pushdown automaton (NIDPDA)* is a tuple $A = (\tilde{\Sigma}, Q, \Gamma, Q_0, \bot, [\delta_s]_{s\in\Sigma_0}, F)$, where $\tilde{\Sigma} = (\Sigma_{-1}, \Sigma_{+1}, \Sigma_0)$ is the input alphabet of left brackets, right brackets, and neutral symbols, Q is a finite set of states, $Q_0, F \subseteq Q$ are the sets of initial and final states, respectively, Γ is the finite stack alphabet with $\bot \notin \Gamma$, δ_s maps Q to 2^Q for each $s \in \Sigma_0$, $\delta_<$ maps Q to $2^{Q\times\Gamma}$ for each $< \in \Sigma_{+1}$, and $\delta_>$ maps $Q \times \Gamma$ to 2^Q. A configuration, a single step computation relation, and its reflexive-transitive closure are defined similarly to the case of DIDPDA; for details, the reader may refer to a recent survey [6]. The language recognized by an NIDPDA is the set of strings that can reach an accepting configuration (with any content of the stack) from the initial configuration.

Every NIDPDA with n states has an equivalent (deterministic) IDPDA with at most 2^{n^2} states [2] and there is a $2^{\Omega(n \log n)}$-state lower bound [1]. The lower bound can actually be improved to match the upper bound, as follows.

Definition 9. *Let the alphabet be* $\Sigma_{+1} = \{<\}$, $\Sigma_{-1} = \{>_R \mid R \subseteq Q \times Q\}$, $\Sigma_0 = \{P \mid P \subseteq Q \times Q\}$. *Define an NIDPDA with the set of states* $Q = \{1, 2, \ldots, n\}$, *with all states initial and accepting, with* $\Gamma = Q$ *as the stack alphabet, and with the following transitions. By the left bracket, the automaton pushes the current state and stays in the same state. By a right bracket marked with* R, *it enters the state 1 if the pair (current state, stack symbol) is in* R, *and has no transitions otherwise. By each* $P \in \Sigma_0$, *the automaton may move from state* i *to any* j *with* $(i, j) \in P$.

Proposition 10. *Let* A *be the NIDPDA from Definition 9 and* $P, R \subseteq Q \times Q$. *Then* $<P>_R \in L(A)$ *if and only if* $P \cap R \neq \varnothing$.

Proof. Let $P \cap R \neq \varnothing$. Then there is a pair (i, j) such that $(i, j) \in P \cap R$. The NIDPDA A starting in the state i, remains in the state i and pushes i on $<$, then on reading P it guesses $j \in P$, and after reading $>_R$ with $(i, j) \in R$ it accepts in the accepting state j.

Now let $P \cap R = \varnothing$. The NIDPDA A starting in an arbitrary state i, remains in the state i and pushes i on $<$, then on reading P it guesses $j \in P$, and upon reading $>_R$ with $(i, j) \notin T$ it rejects by an undefined transition. □

Theorem 11 (Determinization). *Let* A *be the NIDPDA from Definition 9. Then every DIDPDA for* $L(A)$ *requires at least* 2^{n^2} *states.*

Proof. Let $w = <$ and $S = \{P \mid P \subseteq Q \times Q\}$. It is enough to show that every two distinct strings in S can be separated by some string. Let $P \neq R$. Then, without loss of generality, there is a pair (i, j) with $(i, j) \in P \cap \overline{R}$. By Proposition 10, the string $<P>_{\overline{R}}$ is in $L(A)$, while the string $<R>_{\overline{R}}$ is not in $L(A)$. Then, every DIDPDA recognizing $L(A)$ requires at least 2^{n^2} states. □

The following two results are obtained using the same construction. They improve the results of Okhotin and Salomaa [7], who established a lower bound $2^{\Omega(n^2)}$ on the same product in each case.

Theorem 12 (Complement). *Let A be the NIDPDA from Definition 9. Then for every NIDPDA recognizing the complement of $L(A)$, the product of the number of its states by the number of its stack symbols must be at least 2^{n^2}.*

Proof. Let $\mathcal{F} = \{(<P,>_{\overline{P}}) \mid P \subseteq Q \times Q\}$ be a set of pairs of strings of depth 1. To prove the theorem it is enough to show that $\mathcal{F}$ is a fooling set for the complement of $L(A)$. Let $P, R \subseteq Q \times Q$.

(1) By Proposition 10, the string $<P>_{\overline{P}}$ is not in $L(A)$.
(2) If $P \neq R$, then without loss of generality, $P \cap \overline{R} \neq \varnothing$, so by Proposition 10, the string $<P>_{\overline{P}}$ is in $L(A)$.

Hence $\mathcal{F}$ is a fooling set for the complement of $L(A)$, and the theorem follows from a known result [7, Lemma 1]. □

An NIDPDA A is *unambiguous, UIDPDA* if it has exactly one accepting computation on every string in $L(A)$. Since every deterministic IDPDA is unambiguous, every NIDPDA with n states has an equivalent UIDPDA with at most 2^{n^2} and $|\Sigma_{+1}| \cdot 2^{n^2}$ stack symbols [1]. A lower bound $2^{\Omega(n \log n)}$ on the product of the number of its states by the number of its stack symbols was given in [7, Lemma 3]. To get this lower bound, a method based on the rank of some matrices was used. To a set of pairs $F = \{(x_i, y_i)\ i = 1, \ldots, n\}$ of depth k, a matrix $M(F, L)$ is be assigned by settings $M(F, L)_{i,j} = 1$ if $x_i y_j \in L$ and $M(F, L)_{i,j} = 0$ if $x_i y_j \notin L$. Then, the rank of $M(F, L)$ provides a lower bound on $|\Gamma|^k \cdot |Q|$ [7, Lemma 2]. In the next theorem, this bound is improved to 2^{n^2}.

Theorem 13 (NIDPDA to UIDPDA). *Let A be the NIDPDA from Definition 9. Then for every unambiguous IDPDA for $L(A)$, the product of the number of its states by the number of its stack symbols must be at least $2^{n^2} - 1$.*

Proof. Let $F = \{(<P,>_R) \mid P, R \subseteq Q \times Q\}$ be a set of pairs of depth 1. By Proposition 10, $<P>_R \in L$ if and only if $P \cap R \neq \varnothing$. Next, the matrix $M(F, L(A))$ can be viewed as a matrix whose rows and columns are indexed by non-empty subsets of $Q \times Q$ and $M_{S,T} = 1$ if $S \cap T \neq \varnothing$ and $M_{S,T} = 0$ if $S \cap T = \varnothing$. The rank of M is $2^{n^2} - 1$ [4, Lemma 3], and the theorem follows from [7, Lemma 2]. □

Table 2. Updated lower and upper bounds.

	Well-nested inputs		Arbitrary inputs	
	Lower bound	Upper bound	Lower bound	Upper bound
$\cdot$	$\boldsymbol{mn^n}$	$m(n^n + 2^n)$ [8]	$\boldsymbol{mn^n}$	$\boldsymbol{m \cdot (n+1) \cdot n^n \cdot 2^n}$
^{2}	$\boldsymbol{n^n}$	$n(n^n + 2^n)$ [8]	$\boldsymbol{n^n}$	$\boldsymbol{n \cdot (n+1) \cdot n^n \cdot 2^n}$
*	$\boldsymbol{n^n}$	$n^n + 2^n + 1$ [8]	$\boldsymbol{n^n}$	$n^n \cdot 4^n$ [8]
^{R}	$\boldsymbol{n^n}$	n^n [8]	$\boldsymbol{n^n}$	$\boldsymbol{n(n^n - (n-1)^n) + 1}$

6 Conclusion

With the improvements made in this paper, the updated bounds on the state complexity of concatenation, square, star and reversal are presented in Table 2. Some of the bounds are still in want of refinement, and this is recommended as a problem for future research.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: ACM Symposium on Theory of Computing. STOC 2004, Chicago, USA, 13–16 June 2004, pp. 202–211 (2004). https://doi.org/10.1145/1007352.1007390
2. von Braunmühl, B., Verbeek, R.: Input driven languages are recognized in log n space. Ann. Discrete Math. **24**, 1–20 (1985). https://doi.org/10.1016/S0304-0208(08)73072-X
3. Kutrib, M., Malcher, A., Wendlandt, M.: Tinput-driven pushdown automata. In: Durand-Lose, J., Nagy, B. (eds.) MCU 2015. LNCS, vol. 9288, pp. 94–112. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23111-2_7
4. Leung, H.: Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata. SIAM J. Comput. **27**(4), 1073–1082 (1998)
5. Mehlhorn, K.: Pebbling mountain ranges and its application to DCFL-recognition. In: de Bakker, J., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 422–435. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10003-2_89
6. Okhotin, A., Salomaa, K.: Complexity of input-driven pushdown automata. SIGACT News **45**(2), 47–67 (2014). https://doi.org/10.1145/2636805.2636821
7. Okhotin, A., Salomaa, K.: Descriptional complexity of unambiguous input-driven pushdown automata. Theoret. Comput. Sci. **566**, 1–11 (2015). https://doi.org/10.1016/j.tcs.2014.11.015
8. Okhotin, A., Salomaa, K.: State complexity of operations on input-driven pushdown automata. J. Comput. Syst. Sci. **86**, 207–228 (2017). https://doi.org/10.1016/j.jcss.2017.02.001
9. Okhotin, A., Salomaa, K.: Edit distance neighbourhoods of input-driven pushdown automata. In: Weil, P. (ed.) CSR 2017. LNCS, vol. 10304, pp. 260–272. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58747-9_23
10. Okhotin, A., Salomaa, K.: The quotient operation on input-driven pushdown automata. In: Pighizzini, G., Câmpeanu, C. (eds.) DCFS 2017. LNCS, vol. 10316, pp. 299–310. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60252-3_24
11. Okhotin, A., Salomaa, K.: Further closure properties of input-driven pushdown automata. In: Konstantinidis, S., Pighizzini, G. (eds.) DCFS 2018, pp. 224–236. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94631-3_19. to appear
12. Piao, X., Salomaa, K.: Operational state complexity of nested word automata. Theoret. Comput. Sci. **410**, 3290–3302 (2009). https://doi.org/10.1016/j.tcs.2009.05.002
13. Salomaa, K.: Limitations of lower bound methods for deterministic nested word automata. Inf. Comput. **209**, 580–589 (2011). https://doi.org/10.1016/j.ic.2010.11.021